CONSIDERATIONS

- **If a product is added in database, then details of that particular data, we be automatically added in Inventory Table. By default, the value in of Quantity will be 0 and the (last Stock Update) value will be current time of the system.**

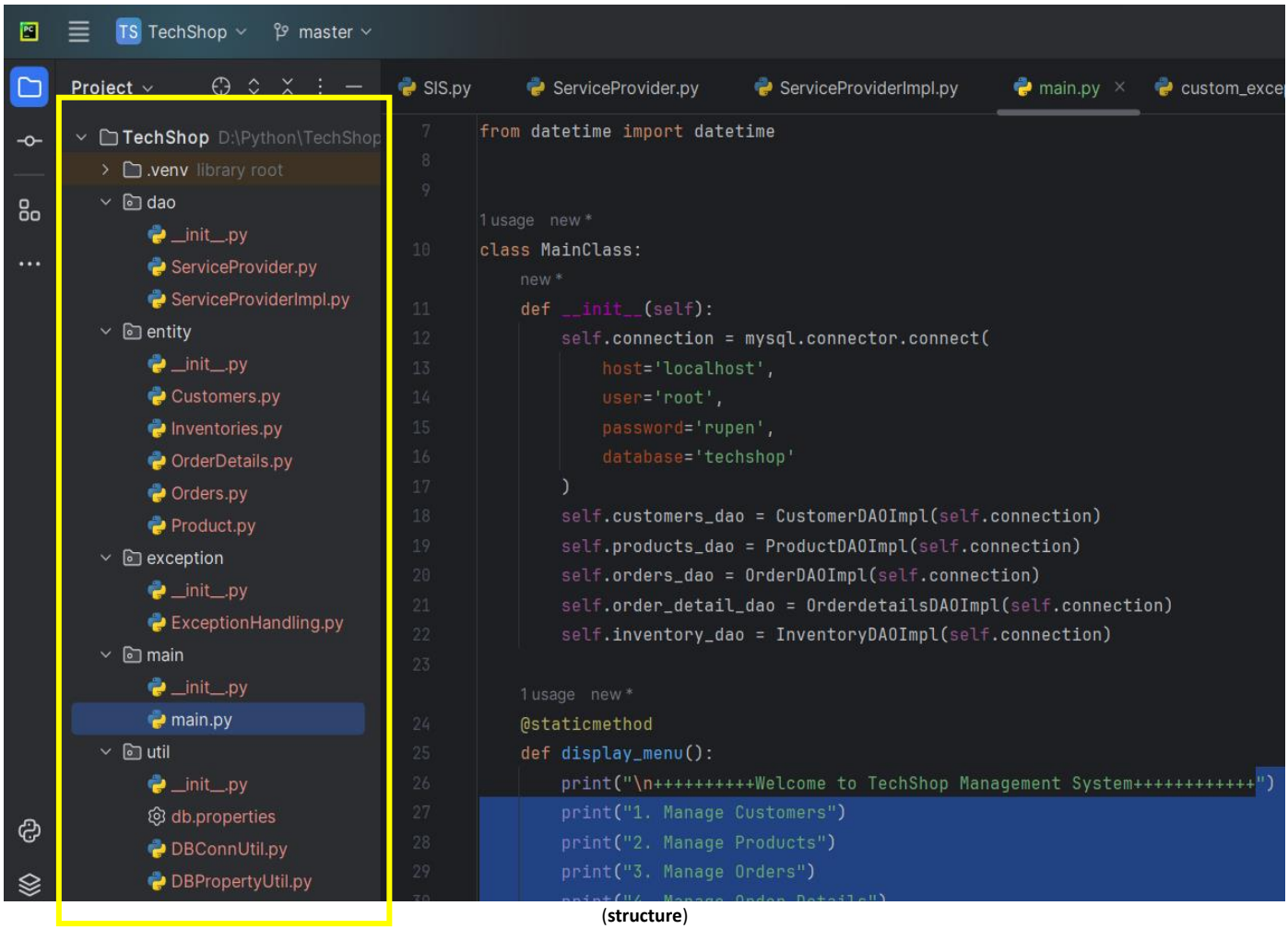
- **If a customer buys a product, that means new data will be created in order table and the data om order details table will be automatically updated like the Quantity, the product_id and the order_id.**

- **If an order is cancelled, the corresponding data associated with that order ID in the order details table will be removed, and the quantity of the product purchased in that order will be added back to the inventory for the same product.**

- **There is no such CREATE or DELETE operations in inventory and order details. Because without an order we can't create order details of any order and similarly we can't create an inventory without adding a new product. Same goes for DELETE operation as well.**

- **As per given structure the ServiceProvider class is an abstract class to showcase functionalities. And ServiceProviderImpl is an implementation class for the abstract class with database interaction.**

- The following Directory structure is to be followed in the application.
  - **entity/model**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider interface/abstract class to showcase functionalities.
    - Create the implementation class for the above interface/abstract class with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **util**
    - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
    - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object.
  - **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.



(structure)

Implement OOPs

Task 1: Classes and Their Attributes:

You are working as a software developer for TechShop, a company that sells electronic gadgets. Your task is to design and implement an application using Object-Oriented Programming (OOP) principles to manage customer information, product details, and orders. Below are the classes you need to create:

Customers Class:

Attributes:

- CustomerID (int)
- FirstName (string)
- LastName (string)
- Email (string)
- Phone (string)
- Address (string)

Products Class:

Attributes:

- ProductID (int)
- ProductName (string)
- Description (string)
- Price (decimal)

Orders Class:

Attributes:

- OrderID (int)
- Customer (Customer) - Use composition
- OrderDate (DateTime)
- TotalAmount (decimal)

OrderDetails Class:

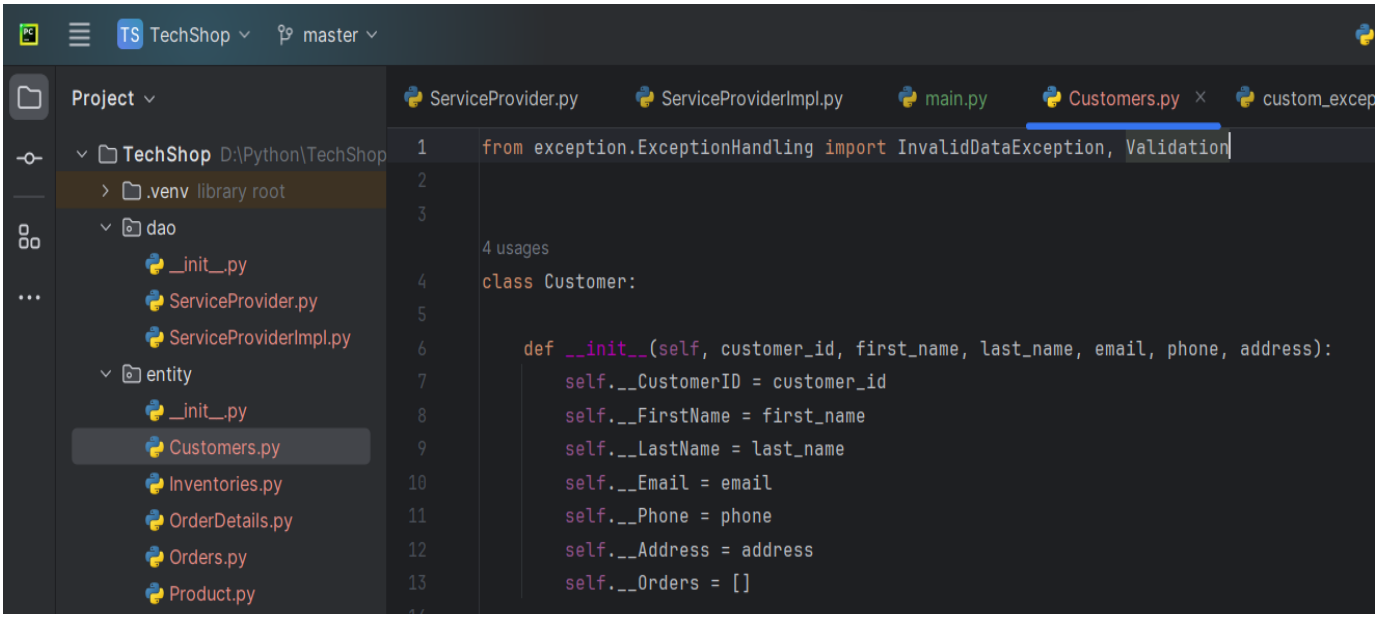
Attributes:

- OrderDetailID (int)
- Order (Order) - Use composition to re
- Product (Product) - Use composition
- Quantity (int)

Inventory class:

Attributes:

- InventoryID(int)
- Product (Composition): The product associated with the inventory item.
- QuantityInStock: The quantity of the product currently in stock.
- LastStockUpdate



Customer class

```python
4 usages
class Product:

    def __init__(self, product_id, name, description, price):
        self.__ProductID = product_id
        self.__ProductName = name
        self.__Description = description
        self.__Price = price
```

Product Class

```python
4 usages
class Order:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.__orderID = order_id
        self.__CustomerID = customer
        self.__orderDate = order_date
        self.__TotalAmount = total_amount
        self.__Products = []
```

Order Class

```python
2 usages
class OrderDetail:
    def __init__(self, order_details_id, order_id, product_id, quantity):
        self.__OrderDetailID = order_details_id
        self.__Order = order_id
        self.__ProductID = product_id
        self.__Quantity = quantity
```

Order Details Class

```python
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
        self.__InventoryID = inventory_id
        self.__Product = product
        self.__QuantityInStock = quantity_in_stock
        self.__LastStockUpdate = last_stock_update
```

Inventory Class

Task 2: Class Creation:

- Create the classes (Customers, Products, Orders, OrderDetails and Inventory) with the specified attributes.
- Implement the constructor for each class to initialize its attributes.
- Implement methods as specified.

A. Customer class:

Methods:

- CalculateTotalOrders(): Calculates the total number of orders placed by this customer.
- GetCustomerDetails(): Retrieves and displays detailed information about the customer.
- UpdateCustomerInfo(): Allows the customer to update their information (e.g., email, phone, or address).

```python

    def calculate_total_orders(self, customer_id):
        pass

    def get_customer_by_id(self, customer_id):
        pass

    def update_customer_info(self, customer_id, new_email, new_phone, new_address):
        pass

```

Customer class (methods)

B. Product class:

Methods:

- GetProductDetails(): Retrieves and displays detailed information about the product.
- UpdateProductInfo(): Allows updates to product details (e.g., price, description).
- IsProductInStock(): Checks if the product is currently in stock.

```python
    def get_product_by_id(self, product_id):
        pass

    def update_product_info(self, price):
        pass

    def is_product_in_stock(self, product_id):
        pass

```

C. Order class:

- CalculateTotalAmount() - Calculate the total amount of the order.
- GetOrderDetails(): Retrieves and displays the details of the order (e.g., product list and quantities).
- UpdateOrderStatus(): Allows updating the status of the order (e.g., processing, shipped).
- CancelOrder(): Cancels the order and adjusts stock levels for products.

```
    def GetOrderDetails(self, order_id):
        pass

    def UpdateOrderStatus(self, order_date):
        pass

    def CalculateTotalAmount(self):
        pass

    def CancelOrder(self):
        pass
```

D. Order Details class:

Methods:

- CalculateSubtotal() - Calculate the subtotal for this order detail.
- GetOrderDetailInfo(): Retrieves and displays information about this order detail.
- UpdateQuantity(): Allows updating the quantity of the product in this order detail.
- AddDiscount(): Applies a discount to this order detail.

```
    def CalculateSubtotal(self):
        pass

    def GetOrderDetailInfo(self):
        pass

    def UpdateQuantity(self):
        pass

    def AddDiscount(self):
        pass

```

E. Inventory class:

Methods:

- GetProduct(): A method to retrieve the product associated with this inventory item.
- GetQuantityInStock(): A method to get the current quantity of the product in stock.
- AddToInventory(int quantity): A method to add a specified quantity of the product to the inventory.
- RemoveFromInventory(int quantity): A method to remove a specified quantity of the product from the inventory.
- UpdateStockQuantity(int newQuantity): A method to update the stock quantity to a new value.
- IsProductAvailable(int quantityToCheck): A method to check if a specified quantity of the product is available in the inventory.
- GetInventoryValue(): A method to calculate the total value of the products in the inventory based on their prices and quantities.
- ListLowStockProducts(int threshold): A method to list products with quantities below a specified threshold, indicating low stock.
- ListOutOfStockProducts(): A method to list products that are out of stock.

```python
    def get_product(self, inventory_id):
        pass

    def get_quantity_in_stock(self, inventory_id):
        pass

    def add_to_inventory(self, inventory_id, quantity):
        pass

    def remove_from_inventory(self, inventory_id, quantity):
        pass

    def update_stock_quantity(self, inventory_id, new_quantity):
        pass

    def is_product_available(self, inventory_id, quantity_to_check):
        pass

    def get_inventory_value(self, inventory_id):
        pass

    def list_low_stock_products(self, threshold):
        pass

    def list_out_of_stock_products(self):
        pass

    def list_all_products(self):
        pass
```

Task 3: Encapsulation:

- Implement encapsulation by making the attributes private and providing public properties (getters and setters) for each attribute.
- Add data validation logic to setter methods (e.g., ensure that prices are non-negative, quantities are positive integers).

```python
@property
def CustomerID(self):
    return self.__CustomerID

3 usages (2 dynamic)
@property
def FirstName(self):
    return self.__FirstName

2 usages (2 dynamic)
@FirstName.setter
def FirstName(self, first_name):
    if isinstance(first_name, str):
        self.__FirstName = first_name
    else:
        raise Exception("First name must be a string.")

3 usages (2 dynamic)
@property
def LastName(self):
    return self.__LastName

2 usages (2 dynamic)
@LastName.setter
def LastName(self, last_name):
    if isinstance(last_name, str):
        self.__LastName = last_name
    else:
        raise Exception("Last name must be a string.")
```

```python
3 usages (2 dynamic)
@property
def Email(self):
    return self.__Email

2 usages (2 dynamic)
@Email.setter
def Email(self, email):
    try:
        Validation.validate_email(email)
        self.__Email = email
    except InvalidDataException as e:
        raise InvalidDataException(str(e))

3 usages (2 dynamic)
@property
def Phone(self):
    return self.__Phone

2 usages (2 dynamic)
@Phone.setter
def Phone(self, phone):
    if isinstance(phone, str):
        self.__Phone = phone
    else:
        raise Exception("Phone must be a string.")

@property
def Orders(self):
    return self.__Orders
```

Customer class (getter & setters)

```python
3 usages (3 dynamic)
@property
def ProductID(self):
    return self.__ProductID

3 usages (2 dynamic)
@property
def ProductName(self):
    return self.__ProductName

2 usages (2 dynamic)
@ProductName.setter
def ProductName(self, product_name):
    if isinstance(product_name, str):
        self.__ProductName = product_name
    else:
        raise Exception("PRODUCT NAME MUST BE STRING")

3 usages (2 dynamic)
@property
def Description(self):
    return self.__Description
```

```python
2 usages (2 dynamic)
@Description.setter
def Description(self, product_description):
    if isinstance(product_description, str):
        self.__Description = product_description
    else:
        raise Exception("DESCRIPTION MUST BE STRING")

3 usages (2 dynamic)
@property
def Price(self):
    return self.__Price

2 usages (2 dynamic)
@Price.setter
def Price(self, product_price):
    if isinstance(product_price, int) and product_price > 0:
        self.__Price = product_price
    else:
        raise Exception("PRICE MUST BE NUMERIC AND NON NEGATIVE")
```

Product class (getter & setters)

```
1 usage (1 dynamic)
@property
def orderID(self):
    return self.__orderID

3 usages (3 dynamic)
@property
def CustomerID(self):
    return self.__CustomerID

3 usages (2 dynamic)
@property
def orderDate(self):
    return self.__orderDate
```

```
2 usages (2 dynamic)
@orderDate.setter
def orderDate(self, order_date):
    if isinstance(order_date, str):
        self.__orderDate = order_date
    else:
        raise Exception("Invalid input")

2 usages (1 dynamic)
@property
def TotalAmount(self):
    return self.__TotalAmount

1 usage (1 dynamic)
@TotalAmount.setter
def TotalAmount(self, total_amount):
    if isinstance(total_amount, (int, float)) and total_amount >= 0:
        self.__TotalAmount = total_amount
    else:
        raise Exception("Must be integer and non negative")

def setProducts(self, products):
    self.__Products = products
```

Order class (getter & setters)

```
@property
def OrderDetailID(self):
    return self.__OrderDetailID

@property
def Order(self):
    return self.__Order

3 usages (3 dynamic)
@property
def ProductID(self):
    return self.__ProductID

3 usages (2 dynamic)
@property
def Quantity(self):
    return self.__Quantity

2 usages (2 dynamic)
@Quantity.setter
def Quantity(self, quantity):
    if quantity > 0:
        self.__Quantity = quantity
    else:
        raise Exception("Quantity must be 0 or greater than 0")
```

Order Details class (getter & setters)

```python
    @property
    def InventoryID(self):
        return self.__InventoryID

    @property
    def Product(self):
        return self.__Product

    1 usage
    @property
    def QuantityInStock(self):
        return self.__QuantityInStock

    @property
    def LastStockUpdate(self):
        return self.__LastStockUpdate

    @QuantityInStock.setter
    def QuantityInStock(self, quantity):
        if quantity >= 0:
            self.__QuantityInStock = quantity
        else:
            raise Exception("Quantity must be a non-negative integer.")
```

Inventory class (getter & setters)

Task 4: Composition:

Ensure that the Order and OrderDetail classes correctly use composition to reference Customer and Product objects.

- Orders Class with Composition:
  - In the Orders class, we want to establish a composition relationship with the Customers class, indicating that each order is associated with a specific customer.
  - In the Orders class, we've added a private attribute customer of type Customers, establishing a composition relationship. The Customer property provides access to the Customers object associated with the order.
- OrderDetails Class with Composition:
  - Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered.
  - In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the product in the order detail.

```python
class Order:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.__orderID = order_id
        self.__CustomerID = customer
        self.__orderDate = order_date
        self.__TotalAmount = total_amount
        self.__Products = []
```

Task 5: Exceptions handling

- **Data Validation:**

```python
2 usages
class Validation:
    1 usage
    @staticmethod
    def validate_email(email):
        if '@' not in email or '.' not in email:
            raise InvalidDataException("Invalid email format")

```

Task 7: Database Connectivity

- Implement a DatabaseConnector class responsible for establishing a connection to the "TechShopDB" database. This class should include methods for opening, closing, and managing database connections.
- Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.

```python
import mysql.connector


class DBConnUtil:
    @staticmethod
    def get_connection(connection_properties: dict):
        try:
            connection = mysql.connector.connect(
                host=connection_properties['host'],
                port=connection_properties['port'],
                user=connection_properties['user'],
                password=connection_properties['password'],
                database=connection_properties['database']
            )
            return connection
        except mysql.connector.Error as e:
            print(f"Error connecting to database: {e}")

```

main.py db.properties DBConnUtil.py DBPropertyUtil.py Inventorie

```python
class DBPropertyUtil:
    @staticmethod
    def get_connection_properties(file_name: str) -> dict:
        connection_properties = {}
        try:
            with open(file_name, 'r') as file:
                for line in file:
                    key, value = line.strip().split('=')
                    connection_properties[key.strip()] = value.strip()
            return connection_properties
        except FileNotFoundError:
            print(f"Error: File {file_name} not found.")
            return {}
```

main.py db.properties DBCon

```
host=localhost
port=3306
user=root
password=pass
database=techshop
```

1: Customer Registration

Description: When a new customer registers on the TechShop website, their information (e.g., name, email, phone) needs to be stored in the database.

Task: Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.

CREATE

```
1 usage
def add_customer(self, customer):
    try:
        cursor = self.connection.cursor()
        sql = ("INSERT INTO Customers (FirstName, LastName, Email, Phone, Address) "
               "VALUES (%s, %s, %s, %s, %s)")
        val = (customer.FirstName, customer.LastName, customer.Email, customer.Phone,
               customer.Address)
        cursor.execute(sql, val)
        self.connection.commit()
        return True
    except mysql.connector.Error as e:
        print(f"Error creating customer: {e}")
        raise DAOException("Error creating customer")
```

READ

```
1 usage
def get_all_customers(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Customers"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

UPDATE

```python
def update_customer_info(self, customer_id, email=None, phone=None, address=None):
    try:
        cursor = self.connection.cursor()
        sql = "UPDATE Customers SET"
        val = []

        if email is not None:
            sql += " Email=%s,"
            val.append(email)

        if phone is not None:
            sql += " Phone=%s,"
            val.append(phone)

        if address is not None:
            sql += " Address=%s,"
            val.append(address)

        # Remove the trailing comma from the SQL query
        if len(val) > 0:
            sql = sql.rstrip(',')
            sql += " WHERE CustomerID=%s"
            val.append(customer_id)

            cursor.execute(sql, val)
            self.connection.commit()
            cursor.close()
            return True
        else:
            print("No parameters provided for update")
            return False
    except mysql.connector.Error as e:
        print(f"Error updating customer: {e}")
```

DELETE

```python
1 usage
def delete_customer(self, customer_id):
    try:
        cursor = self.connection.cursor()
        sql = "DELETE FROM Customers WHERE CustomerID=%s"
        cursor.execute(sql, (customer_id,))
        self.connection.commit()
        cursor.close()
        return True
    except mysql.connector.Error as e:
        print(f"Error deleting customer: {e}")
        raise DAOException("Error deleting customer")
```

2: Product Catalog Management

Description: TechShop regularly updates its product catalog with new items and changes in product details (e.g., price, description). These changes need to be reflected in the database.

Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency.

CREATE

```
1 usage
def add_products(self, product):
    try:
        cursor = self.connection.cursor()
        sql = "INSERT INTO Products (ProductName, Description, Price) VALUES (%s, %s, %s)"
        val = (product.ProductName, product.Description, product.Price)
        cursor.execute(sql, val)
        self.connection.commit()
        product_id = cursor.lastrowid
        sql_insert_inventory = ("INSERT INTO Inventory (ProductID, QuantityInStock, LastStockUpdate) "
                                "VALUES (%s, %s, NOW())")
        inventory_data = (product_id, 0)
        cursor.execute(sql_insert_inventory, inventory_data)
        self.connection.commit()
        return True
    except mysql.connector.Error as e:
        print(f"Error creating customer: {e}")
        raise DAOException("Error creating customer")
```

READ

```
1 usage
def get_all_products(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Products"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

UPDATE

```
1 usage
def update_product_info(self, product_id, new_price=None):
    try:
        cursor = self.connection.cursor()
        sql = "UPDATE Products SET"
        val = []

        if new_price is not None:
            sql += " Price=%s,"
            val.append(new_price)

        if len(val) > 0:
            sql = sql.rstrip(',')
            sql += " WHERE ProductID=%s"
            val.append(product_id)

            cursor.execute(sql, val)
            self.connection.commit()
            cursor.close()
            return True
        else:
            print("No parameters provided for update")
            return False
    except mysql.connector.Error as e:
        print(f"Error updating customer: {e}")
        raise DAOException("Error updating customer")
```

DELETE

```
1 usage
def delete_products(self, product_id):
    try:
        cursor = self.connection.cursor()
        sql = "DELETE FROM Products WHERE ProductID = %s"
        cursor.execute(sql, (product_id,))
        self.connection.commit()
        cursor.close()
        return True
    except mysql.connector.Error as e:
        print(f"Error deleting product: {e}")
        raise DAOException("Error deleting product")
```

3: Placing Customer Orders

Description: Customers browse the product catalog and place orders for products they want to purchase. The orders need to be stored in the database.

Task: Implement an order processing system. Use database connectivity to record customer orders, update product quantities in inventory, and calculate order totals.

CREATE

```
1 usage
def create_orders(self, order, order_details):
    try:
        cursor = self.connection.cursor()
        total_amount = 0
        for order_detail in order_details:
            sql_get_price = "SELECT Price FROM Products WHERE ProductID = %s"
            cursor.execute(sql_get_price, (order_detail.ProductID,))
            price = cursor.fetchone()[0]
            total_amount += order_detail.Quantity * price
        sql_insert_order = "INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES (%s, %s, %s)"
        order_data = (order.CustomerID, order.orderDate, total_amount)
        cursor.execute(sql_insert_order, order_data)
        order_id = cursor.lastrowid
        for order_detail in order_details:
            sql_insert_order_detail = "INSERT INTO OrderDetails (OrderID, ProductID, Quantity) VALUES (%s, %s, %s)"
            order_detail_data = (order_id, order_detail.ProductID, order_detail.Quantity)
            cursor.execute(sql_insert_order_detail, order_detail_data)
        self.connection.commit()
        return True
    except mysql.connector.Error as e:
        print(f"Error creating customer: {e}")
        raise DAOException("Error adding order")
```

READ

```
1 usage
def display_orders(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Orders"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

UPDATE

```
1 usage
def UpdateOrderStatus(self, order_id):
    try:
        cursor = self.connection.cursor()
        sql_get_order_date = "SELECT OrderDate FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_get_order_date, (order_id,))
        result = cursor.fetchone()
        if result:
            order_date = result[0]
            current_date = datetime.now()
            order_date = datetime.combine(order_date, datetime.min.time())
            difference = current_date - order_date
            if difference.days > 3:
                return "shipped"
            else:
                return "Processing"
        else:
            return "Order not found"
    except mysql.connector.Error as e:
        print(f"Error updating order status: {e}")
        raise DAOException("Error updating order status")
```

DELETE

```
1 usage
def CancelOrder(self, order_id):
    try:
        cursor = self.connection.cursor()
        sql_select_order = "SELECT OrderID FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_select_order, (order_id,))
        result = cursor.fetchone()
        if not result:
            raise DAOException("Order not found")
        sql_select_order_details = "SELECT ProductID, Quantity FROM OrderDetails WHERE OrderID = %s"
        cursor.execute(sql_select_order_details, (order_id,))
        order_details = cursor.fetchall()
        for product_id, quantity in order_details:
            sql_update_inventory = ("UPDATE Inventory SET QuantityInStock ="
                                    " QuantityInStock + %s WHERE ProductID = %s")
            cursor.execute(sql_update_inventory, (quantity, product_id))
        sql_delete_order_details = "DELETE FROM OrderDetails WHERE OrderID = %s"
        cursor.execute(sql_delete_order_details, (order_id,))
        sql_delete_order = "DELETE FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_delete_order, (order_id,))
        self.connection.commit()
        cursor.close()
        return True
    except mysql.connector.Error as e:
        print(f"Error deleting product: {e}")
        raise DAOException("Error deleting product")
```

4: Tracking Order Status

Description: Customers and employees need to track the status of their orders. The order status information is stored in the database.

Task: Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

ORDER DETAILS OF EVERY CUSTOMER

```
1 usage
def GetAllOrderDetail(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM orderdetails"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

ORDER DETAILS WITH PRODUCT INFORMATION

```
def GetOrderDetailInfo(self, order_detail_id):
    try:
        cursor = self.connection.cursor()
        sql = ("SELECT od.OrderDetailID, o.OrderID, p.ProductName, od.Quantity, p.Price"
               " FROM OrderDetails od INNER JOIN Orders o ON od.OrderID = o.OrderID "
               "INNER JOIN Products p ON od.ProductID = p.ProductID WHERE od.OrderDetailID = %s")
        cursor.execute(sql, (order_detail_id,))
        result = cursor.fetchone()
        if result:
            order_detail_id, order_id, product_name, quantity, price = result
            print("Order Detail ID:", order_detail_id)
            print("Order ID:", order_id)
            print("Product Name:", product_name)
            print("Quantity:", quantity)
            print("Price:", price)
        else:
            print("Order detail not found.")
    except mysql.connector.Error as e:
        print(f"Error getting order detail info: {e}")
        raise DAOException("Error getting order detail info")
```

UPDATE QUANTITY OF PRODUCT TO BE ORDERED

```
def UpdateQuantity(self, order_detail_id, new_quantity):
    try:
        cursor = self.connection.cursor()
        sql_get_price = ("SELECT p.Price FROM OrderDetails od JOIN Products p ON od.ProductID = "
                         "p.ProductID WHERE od.OrderDetailID = %s")
        cursor.execute(sql_get_price, (order_detail_id,))
        """price = cursor.fetchone()[0]"""
        sql_update_quantity = "UPDATE OrderDetails SET Quantity = %s WHERE OrderDetailID = %s"
        cursor.execute(sql_update_quantity, (new_quantity, order_detail_id))

        sql_update_total_amount = ("UPDATE Orders SET TotalAmount = (SELECT SUM(od.Quantity * p.Price) "
                                   "FROM OrderDetails od JOIN Products p ON od.ProductID = p.ProductID "
                                   "WHERE od.OrderID = (SELECT OrderID FROM OrderDetails "
                                   "WHERE OrderDetailID = %s)) WHERE OrderID = "
                                   "(SELECT OrderID FROM OrderDetails WHERE OrderDetailID = %s)")
        cursor.execute(sql_update_total_amount, (order_detail_id, order_detail_id))

        self.connection.commit()
        print("Quantity updated successfully.")
    except mysql.connector.Error as e:
        print(f"Error updating quantity: {e}")
        raise DAOException("Error updating quantity")
```

5: Inventory Management

Description: TechShop needs to manage product inventory, including adding new products, updating stock levels, and removing discontinued items.

Task: Create an inventory management system with database connectivity. Implement features for adding new products, updating quantities, and handling discontinued products.

ADD QUANTITY

```
1 usage
def add_to_inventory(self, inventory_id, quantity):
    try:
        cursor = self.connection.cursor()
        sql = "UPDATE Inventory SET QuantityInStock = QuantityInStock + %s WHERE InventoryID = %s"
        cursor.execute(sql, (quantity, inventory_id))
        self.connection.commit()
        cursor.close()
        print("Quantity added to inventory successfully.")
    except mysql.connector.Error as e:
        print(f"Error adding to inventory: {e}")
        raise DAOException("Error adding to inventory")
```

REMOVE QUANTITY

```
1 usage
def remove_from_inventory(self, inventory_id, quantity):
    try:
        cursor = self.connection.cursor()
        sql = "UPDATE Inventory SET QuantityInStock = QuantityInStock - %s WHERE InventoryID = %s"
        cursor.execute(sql, (quantity, inventory_id))
        self.connection.commit()
        cursor.close()
        print("Quantity removed from inventory successfully.")
    except mysql.connector.Error as e:
        print(f"Error removing from inventory: {e}")
        raise DAOException("Error removing from inventory")
```

DISPLAY ALL PRODUCTS

```
def list_all_products(self):
    try:
        cursor = self.connection.cursor()
        sql = ("SELECT p.ProductID, p.ProductName, p.Description, p.Price, i.QuantityInStock "
               "FROM Products p JOIN Inventory i ON p.ProductID = i.ProductID")
        cursor.execute(sql)
        results = cursor.fetchall()
        cursor.close()
        if results:
            print("All Products in Inventory:")
            for result in results:
                print(
                    f"Product ID: {result[0]}, Name: {result[1]}, Description: {result[2]}, Price: {result[3]},"
                    f" Quantity in Stock: {result[4]}")
        else:
            print("No products found in the inventory.")
    except mysql.connector.Error as e:
        print(f"Error listing all products: {e}")
        raise DAOException("Error listing all products")
```

6: Sales Reporting

Description: TechShop management requires sales reports for business analysis. The sales data is stored in the database.

Task: Design and implement a reporting system that retrieves sales data from the database and generates reports based on specified criteria.

INVENTORY VALUE

```
1 usage
def get_inventory_value(self, inventory_id):
    try:
        cursor = self.connection.cursor()
        sql = ("SELECT SUM(p.Price * i.QuantityInStock) "
               "FROM Products p JOIN Inventory i ON p.ProductID = i.ProductID "
               "WHERE i.InventoryID = %s")
        cursor.execute(sql, (inventory_id,))
        result = cursor.fetchone()
        cursor.close()
        if result:
            return result[0]
        else:
            print("Inventory not found.")
            return None
    except mysql.connector.Error as e:
        print(f"Error calculating inventory value: {e}")
        raise DAOException("Error calculating inventory value")
```

LIST LOW STOCK PRODUCTS

```
1 usage
def list_low_stock_products(self, threshold):
    try:
        cursor = self.connection.cursor()
        sql = ("SELECT p.ProductName, i.QuantityInStock "
               "FROM Products p JOIN Inventory i ON p.ProductID = i.ProductID "
               "WHERE i.QuantityInStock < %s")
        cursor.execute(sql, (threshold,))
        results = cursor.fetchall()
        cursor.close()
        if results:
            print("Low Stock Products:")
            for result in results:
                print(f"Product: {result[0]}, Quantity: {result[1]}")
        else:
            print("No low stock products.")
    except mysql.connector.Error as e:
        print(f"Error listing low stock products: {e}")
        raise DAOException("Error listing low stock products")
```

OUT OF STOCK PRODUCTS

```
1 usage
def list_out_of_stock_products(self):
    try:
        cursor = self.connection.cursor()
        sql = ("SELECT p.ProductName "
               "FROM Products p JOIN Inventory i ON p.ProductID = i.ProductID "
               "WHERE i.QuantityInStock <= 0")
        cursor.execute(sql)
        results = cursor.fetchall()
        cursor.close()
        if results:
            print("Out of Stock Products:")
            for result in results:
                print(result[0])
        else:
            print("No out of stock products.")
    except mysql.connector.Error as e:
        print(f"Error listing out of stock products: {e}")
        raise DAOException("Error listing out of stock products")
```

ADD DISCOUNT

```
1 usage
def AddDiscount(self, order_detail_id, discount_percentage):
    try:
        cursor = self.connection.cursor()
        sql_get_subtotal = ("SELECT Quantity * Price FROM OrderDetails od JOIN Products p ON od.ProductID "
                            "= p.ProductID WHERE OrderDetailID = %s")
        cursor.execute(sql_get_subtotal, (order_detail_id,))
        subtotal = cursor.fetchone()[0]
        discount_amount = subtotal * (Decimal(discount_percentage) / 100)
        discounted_subtotal = subtotal - discount_amount
        sql_update_total_amount = ("UPDATE Orders SET TotalAmount = TotalAmount - %s WHERE OrderID "
                                   "= (SELECT OrderID FROM OrderDetails WHERE OrderDetailID = %s)")
        cursor.execute(sql_update_total_amount, (discounted_subtotal, order_detail_id))
        self.connection.commit()
        print("Discount applied successfully.")
    except mysql.connector.Error as e:
        print(f"Error adding discount: {e}")
        raise DAOException("Error adding discount")
```

CALCULATE SUBTOTAL OF ALL PRODUCT BOUGHT BY A CUSTOMER

```
1 usage
def CalculateSubtotal(self, order_detail_id):
    try:
        cursor = self.connection.cursor()
        sql = ("SELECT p.Price, od.Quantity FROM Products p INNER JOIN OrderDetails od ON p.ProductID ="
               " od.ProductID WHERE od.OrderDetailID = %s")
        cursor.execute(sql, (order_detail_id,))
        result = cursor.fetchone()
        if result:
            price, quantity = result
            subtotal = price * quantity
            return subtotal
        else:
            return None
    except mysql.connector.Error as e:
        print(f"Error calculating subtotal: {e}")
        raise DAOException("Error calculating subtotal")
```

CHECK PRODUCT IS SHIPPED OR NOT

```
1 usage
def UpdateOrderStatus(self, order_id):
    try:
        cursor = self.connection.cursor()
        sql_get_order_date = "SELECT OrderDate FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_get_order_date, (order_id,))
        result = cursor.fetchone()
        if result:
            order_date = result[0]
            current_date = datetime.now()
            order_date = datetime.combine(order_date, datetime.min.time())
            difference = current_date - order_date
            if difference.days > 3:
                return "shipped"
            else:
                return "Processing"
        else:
            return "Order not found"
    except mysql.connector.Error as e:
        print(f"Error updating order status: {e}")
        raise DAOException("Error updating order status")
```

CALCULATE TOTAL AMOUNT OF TRANSACTION

```
1 usage
def CalculateTotalAmount(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT SUM(TotalAmount) from Orders"
        cursor.execute(sql)
        total_price_info = cursor.fetchone()
        cursor.close()
        if total_price_info:
            return total_price_info
        else:
            return 0
    except mysql.connector.Error as e:
        print(f"Error calculating total amount for order: {e}")
        raise DAOException("Error calculating total amount for order")
```

GET PRODUCT IN STOCK

```
1 usage
def is_product_in_stock(self, product_id):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT QuantityInStock FROM Inventory WHERE ProductID = %s"
        cursor.execute(sql, (product_id,))
        total_orders = cursor.fetchone()
        cursor.close()
        if total_orders is not None:
            return True
        else:
            return False
    except mysql.connector.Error as e:
        print(f"Error calculating total orders: {e}")
        raise DAOException("Error calculating total orders")
```

UPDATE STOCK QUANTITY

```
1 usage
def update_stock_quantity(self, inventory_id, new_quantity):
    try:
        cursor = self.connection.cursor()
        sql = "UPDATE Inventory SET QuantityInStock = %s WHERE InventoryID = %s"
        cursor.execute(sql, (new_quantity, inventory_id))
        self.connection.commit()
        cursor.close()
        print("Stock quantity updated successfully.")
    except mysql.connector.Error as e:
        print(f"Error updating stock quantity: {e}")
        raise DAOException("Error updating stock quantity")
```

HOW DOES THIS APPLICATION WORK

```
++++++++++Welcome to TechShop Management System+++++++++++++
1. Manage Customers
2. Manage Products
3. Manage Orders
4. Manage Order Details
5. Manage Inventory
6. Exit
Enter your choice: 1

+++++++++++++++++CUSTOMER MENU++++++++++++++++++

Menu:
1. Add Customer(C)
2. Get all Customer Details(R)
3. Update Customer Information(U)
4. Delete Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
```

```
Menu:
1. Add Customer(C)
2. Get all Customer Details(R)
3. Update Customer Information(U)
4. Delete Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 1
Enter First Name: Biswajit
Enter Last Name: Roy
Enter Email: roy_biswajit@gmail.com
Enter Phone: 6891756491
Enter Address: 53/2 VT road
Customer added successfully.
```

```
Menu:
1. Add Customer(C)
2. Get all Customer Details(R)
3. Update Customer Information(U)
4. Delete Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 2
Customer ID: 11 || Name: Aarav Patel || Email: aarav.patel@example.com || Phone: 9876543210 || Address: 456 Tulsi Lane
Customer ID: 12 || Name: Aditi Sharma || Email: aditi.sharma@example.com || Phone: 7778889999 || Address: 789 Ganges Street
Customer ID: 13 || Name: Arjun Verma || Email: arjun.verma@example.com || Phone: 5554443333 || Address: 234 Himalaya Road
Customer ID: 14 || Name: Avni Gupta || Email: new.email@example.com || Phone: 1112220000 || Address: 123 New Street
Customer ID: 15 || Name: Rahul Singh || Email: rahul.singh@example.com || Phone: 6667778888 || Address: 890 Mango Lane
Customer ID: 16 || Name: Neha Yadav || Email: neha.yadav@example.com || Phone: 1112223333 || Address: 123 Rose Garden
Customer ID: 17 || Name: Vedant Kumar || Email: vedant.kumar@example.com || Phone: 9998887777 || Address: 678 Lotus Street
Customer ID: 18 || Name: Isha Shah || Email: isha.shah@example.com || Phone: 3332221111 || Address: 345 Peacock Lane
Customer ID: 19 || Name: Kabir Joshi || Email: kabir.joshi@example.com || Phone: 8889990000 || Address: 456 Mango Avenue
Customer ID: 20 || Name: Ananya Das || Email: ananya.das@example.com || Phone: 4445556666 || Address: 789 Jasmine Road
Customer ID: 21 || Name: Vaybhav Sharma || Email: vaybhav.sharma@example.com || Phone: 1234567890 || Address: 123 Nari Colony
Customer ID: 24 || Name: Biswarup Roy || Email: roy_biswayan@example.com || Phone: 8671489137 || Address: 91 Pally Burdwan
Customer ID: 25 || Name: Biswayan Roy || Email: roy_biswayan@gmail.com || Phone: 7892546317 || Address: 56 Makhla
Customer ID: 26 || Name: Arundhati Roy || Email: roy_arundhati@example.com || Phone: 7581943789 || Address: 82 Pushkar
Customer ID: 28 || Name: Biswajit Roy || Email: roy_biswajit@gmail.com || Phone: 6891756491 || Address: 53/2 VT road
```

```
Menu:
1. Add Customer(C)
2. Get all Customer Details(R)
3. Update Customer Information(U)
4. Delete Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 6
Enter Customer ID: 16
Total orders for customer 16: 1
```

```
12 •  SELECT *
13    FROM customers
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	Email	Phone	Address
12	Aditi	Sharma	aditi.sharma@example.com	7778889999	789 Ganges Street
13	Arjun	Verma	arjun.verma@example.com	5554443333	234 Himalaya Road
14	Avni	Gupta	new.email@example.com	1112220000	123 New Street
15	Rahul	Singh	rahul.singh@example.com	6667778888	890 Mango Lane
16	Neha	Yadav	neha.yadav@example.com	1112223333	123 Rose Garden
17	Vedant	Kumar	vedant.kumar@example.com	9998887777	678 Lotus Street
18	Isha	Shah	isha.shah@example.com	3332221111	345 Peacock Lane
19	Kabir	Joshi	kabir.joshi@example.com	8889990000	456 Mango Avenue
20	Ananya	Das	ananya.das@example.com	4445556666	789 Jasmine Road
21	Vaybhav	Sharma	vaybhav.sharma@example....	1234567890	123 Nari Colony
24	Biswarup	Roy	roy_biswayan@example.com	8671489137	91 Pally Burdwan
25	Biswayan	Roy	roy_biswayan@gmail.com	7892546317	56 Makhla
26	Arundhati	Roy	roy_arundhati@example.com	7581943789	82 Pushkar
28	Biswajit	Roy	roy_biswajit@gmail.com	6891756491	53/2 VT road
NULL	NULL	NULL	NULL	NULL	NULL

customers 1

```
Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 2
Product ID: 11 || Name: Smartphone || Description: Electronic Gadgets || Price: 879.99
Product ID: 12 || Name: Air Conditioner || Description: Home Appliances || Price: 549.99
Product ID: 13 || Name: Smart Refrigerator || Description: Electronic Gadgets || Price: 989.99
Product ID: 14 || Name: Smart Washing Machine || Description: Electronic Gadgets || Price: 43.99
Product ID: 15 || Name: Convection Microwave Oven || Description: Kitchen Appliances || Price: 141.56
Product ID: 16 || Name: Laptop || Description: Electronic Gadgets || Price: 500.65
Product ID: 17 || Name: Bluetooth Speaker || Description: Electronic Gadgets || Price: 87.11
Product ID: 18 || Name: Robotic Vacuum Cleaner || Description: Electronic Gadgets || Price: 163.34
Product ID: 19 || Name: Steam Iron || Description: Home Appliances || Price: 32.66
Product ID: 20 || Name: Professional Hair Dryer || Description: Beauty Appliances || Price: 43.55
Product ID: 21 || Name: Smartwatch || Description: Electronic Gadgets || Price: 149.99
```

```
Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 1
Enter Product Name: Mouse
Enter the Description: Electronice Gadgets
Enter the Price: 365.23
Product added successfully.
```

```
12 •  SELECT *
13    FROM products
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

ProductID	ProductName	Description	Price
11	Smartphone	Electronic Gadgets	879.99
12	Air Conditioner	Home Appliances	549.99
13	Smart Refrigerator	Electronic Gadgets	989.99
14	Smart Washing Machine	Electronic Gadgets	43.99
15	Convection Microwave Oven	Kitchen Appliances	141.56
16	Laptop	Electronic Gadgets	500.65
17	Bluetooth Speaker	Electronic Gadgets	87.11
18	Robotic Vacuum Cleaner	Electronic Gadgets	163.34
19	Steam Iron	Home Appliances	32.66
20	Professional Hair Dryer	Beauty Appliances	43.55
21	Smartwatch	Electronic Gadgets	149.99
23	Mouse	Electronice Gadgets	365.23
NULL	NULL	NULL	NULL

```
+++++++++++++++++ORDER MENU+++++++++++++++++++

Menu:
1. Create Order(C)
2. Display Orders(R)
3  Cancel Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/ shipped)
7. Exit
Enter your choice (1-8): 2
Order ID: 111 || Customer ID: 11 || Order Date: 2024-02-01 || Total Amount: 1044.99
Order ID: 112 || Customer ID: 12 || Order Date: 2024-02-02 || Total Amount: 989.99
Order ID: 113 || Customer ID: 13 || Order Date: 2024-02-03 || Total Amount: 2287.96
Order ID: 115 || Customer ID: 15 || Order Date: 2024-02-05 || Total Amount: 87.99
Order ID: 116 || Customer ID: 16 || Order Date: 2024-02-06 || Total Amount: 329.98
Order ID: 117 || Customer ID: 17 || Order Date: 2024-02-07 || Total Amount: 97.98
Order ID: 119 || Customer ID: 19 || Order Date: 2024-02-09 || Total Amount: 600.00
Order ID: 120 || Customer ID: 20 || Order Date: 2024-02-10 || Total Amount: 950.00
Order ID: 122 || Customer ID: 25 || Order Date: 2024-02-04 || Total Amount: 1099.98
```

```
Menu:
1. Create Order(C)
2. Display Orders(R)
3  Cancel Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/ shipped)
7. Exit
Enter your choice (1-8): 6
Enter Order ID: 111
Processing
```

```
Menu:
1. Create Order(C)
2. Display Orders(R)
3  Cancel Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/ shipped)
7. Exit
Enter your choice (1-8): 1
Enter Customer id: 28
Enter Product ID: 21
Enter Quantity: 2
Add more products? (yes/no): no
Order added successfully.
```

```
12 •  SELECT *
13    FROM orders
```

Result Grid | Filter Rows: | Edit:

	OrderID	CustomerID	OrderDate	TotalAmount
▶	111	11	2024-02-01	1044.99
	112	12	2024-02-02	989.99
	113	13	2024-02-03	2287.96
	115	15	2024-02-05	87.99
	116	16	2024-02-06	329.98
	117	17	2024-02-07	97.98
	119	19	2024-02-09	600.00
	120	20	2024-02-10	950.00
	122	25	2024-02-04	1099.98
	123	28	2024-02-04	299.98
•	NULL	NULL	NULL	NULL

```
+++++++++++++++++ORDER DETAILS MENU+++++++++++++++++++

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8): 5
Order Detail ID:11 || Order ID:111 || Product ID: 11 ||Quantity: 1
Order Detail ID:12 || Order ID:111 || Product ID: 12 ||Quantity: 2
Order Detail ID:13 || Order ID:112 || Product ID: 13 ||Quantity: 1
Order Detail ID:14 || Order ID:113 || Product ID: 14 ||Quantity: 3
Order Detail ID:15 || Order ID:113 || Product ID: 15 ||Quantity: 1
Order Detail ID:17 || Order ID:115 || Product ID: 17 ||Quantity: 1
Order Detail ID:18 || Order ID:116 || Product ID: 18 ||Quantity: 2
Order Detail ID:19 || Order ID:117 || Product ID: 19 ||Quantity: 3
Order Detail ID:21 || Order ID:122 || Product ID: 12 ||Quantity: 2
Order Detail ID:22 || Order ID:123 || Product ID: 21 ||Quantity: 2
```

```
Menu:
0. Show Inventory Table
1. List all products in inventory
2. Get product details
3. Add product to inventory
4. Remove product from inventory
5. Update stock quantity
6. Check product availability
7. Calculate inventory value
8. List low stock products
9. List out-of-stock products
A. Exit
Enter your choice: 0
Inventory ID: 11 || Product Name: 11 || Description: 8 || Quantity in Stock: 2024-02-01
Inventory ID: 12 || Product Name: 12 || Description: 15 || Quantity in Stock: 2024-02-02
Inventory ID: 13 || Product Name: 13 || Description: 10 || Quantity in Stock: 2024-02-03
Inventory ID: 14 || Product Name: 14 || Description: 5 || Quantity in Stock: 2024-02-04
Inventory ID: 15 || Product Name: 15 || Description: 12 || Quantity in Stock: 2024-02-05
Inventory ID: 16 || Product Name: 16 || Description: 7 || Quantity in Stock: 2024-02-06
Inventory ID: 17 || Product Name: 17 || Description: 10 || Quantity in Stock: 2024-02-07
Inventory ID: 18 || Product Name: 18 || Description: 4 || Quantity in Stock: 2024-02-08
Inventory ID: 19 || Product Name: 19 || Description: 18 || Quantity in Stock: 2024-02-09
Inventory ID: 20 || Product Name: 20 || Description: 6 || Quantity in Stock: 2024-02-10
Inventory ID: 21 || Product Name: 23 || Description: 0 || Quantity in Stock: 2024-02-04
```

```
Menu:
0. Show Inventory Table
1. List all products in inventory
2. Get product details
3. Add product to inventory
4. Remove product from inventory
5. Update stock quantity
6. Check product availability
7. Calculate inventory value
8. List low stock products
9. List out-of-stock products
A. Exit
Enter your choice: 3
Enter Inventory ID: 21
Enter quantity to add: 5
Quantity added to inventory successfully.
```

```
SELECT *
FROM inventory
```

Result Grid | Filter Rows: | Edit: | Export/I

InventoryID	ProductID	QuantityInStock	LastStockUpdate
11	11	8	2024-02-01
12	12	15	2024-02-02
13	13	10	2024-02-03
14	14	5	2024-02-04
15	15	12	2024-02-05
16	16	7	2024-02-06
17	17	10	2024-02-07
18	18	4	2024-02-08
19	19	18	2024-02-09
20	20	6	2024-02-10
21	23	5	2024-02-04
NULL	NULL	NULL	NULL

```
++++++++++Welcome to TechShop Management System+++++++++++
1. Manage Customers
2. Manage Products
3. Manage Orders
4. Manage Order Details
5. Manage Inventory
6. Exit
Enter your choice: 6
Exiting the program. Goodbye!

Process finished with exit code 0
```

---- THANK YOU ----